Revision 1.2.3 as of 2015-06-10
Firmware 1 (build 4831), Driver v1.0.0
cronologic GmbH & Co. KG

cronologic

Ndigo250M-14
Ndigo125M-14

User Guide

Contents

1 Introduction **1**
1.1 Features . . . 1

2 Hardware **3**
2.1 Installing the Board . . . 3
2.2 External Inputs and Connectors . . . 5
2.2.1 Connectors . . . 5
2.2.2 Analog Inputs . . . 6
2.2.3 Digital Inputs . . . 7
2.3 Extension Card . . . 8
2.4 Ndigo250M-14 Functionality . . . 9
2.4.1 Zero Suppression . . . 9
2.4.2 Trigger Blocks . . . 10
2.4.3 Gating Blocks . . . 14
2.4.4 Auto Triggering Function Generator . . . 16
2.4.5 Timestamp Channel . . . 17
2.4.6 Data Lookup Table . . . 17
2.5 Multiple Ndigo boards synchronization . . . 18
2.5.1 Calibration Procedure . . . 19
2.5.2 Synchronizing with an HPTDC8-PCI . . . 21
2.6 Performing a firmware update . . . 22
2.7 Calibrating the TDC . . . 23

3 Driver Programming API **25**
3.1 Constants . . . 25
3.2 Initialization . . . 26
3.2.1 Structure ndigo_init_parameters . . . 26
3.3 Status Information . . . 28
3.3.1 Functions for Information Retrieval . . . 28
3.3.2 Structure ndigo_static_info . . . 28
3.3.3 Structure ndigo_param_info . . . 30
3.3.4 Structure ndigo_fast_info . . . 30
3.3.5 Structure ndigo_slow_info . . . 31
3.4 Configuration . . . 32
3.4.1 Structure ndigo250m_configuration . . . 32
3.4.2 Structure ndigo_trigger . . . 34

3.4.3 Structure ndigo_trigger_block . . . 35
3.4.4 Structure ndigo_gating_block . . . 36
3.4.5 Structure ndigo_extension_block . . . 38
3.4.6 Run Time Control . . . 38
3.5 Readout . . . 39
3.5.1 Input Structure ndigo250m_read_in . . . 39
3.5.2 Input Structure ndigo250m_read_out . . . 39
3.6 Other Functions . . . 41
3.6.1 LED control . . . 41

4 Packet Format **43**
4.0.2 Output Structure ndigo_packet . . . 43

5 C Example **45**

6 Technical Data **49**
6.1 Operating Conditions . . . 49
6.1.1 Environmental Conditions for Operation . . . 49
6.1.2 Environmental Conditions for Storage . . . 49
6.2 Digitizer Characteristics . . . 49
6.3 TDC Characteristics . . . 50
6.4 Electrical Characteristics . . . 50
6.4.1 Oscillator . . . 50
6.4.2 Power Supply . . . 50
6.4.3 Analog Inputs . . . 50
6.4.4 Digital Inputs . . . 51
6.5 Information Required by DIN EN 61010-1 . . . 52
6.5.1 Manufacturer . . . 52
6.5.2 Intended Use and System Integration . . . 52
6.5.3 Cooling . . . 52
6.5.4 Environmental Conditions . . . 53
6.5.5 Inputs . . . 53
6.5.6 Recycling . . . 53

1 Introduction

The Ndigo250M and Ndigo125M are digitizer and transient recorders designed to sample relatively shorts pulses in rapid repetition. They produce a stream of output packets, each containing data from a single trigger event together with a timestamp.

1.1 Features

- 14 bit dynamic range
- up to 250 Msps sample rate
- 4 channels
- digital input with TDC that can also be used for gating and triggering
- 2nd digital input for gating or triggering
- PCIe x8 Gen 1 with 1400 MB/s throughput
- multiple boards can be synchronized

The Ndigo125M and Ndigo250M are almost identical. The only difference is the use of a slower ADC in the Ndigo125M. If not otherwise noted this user guide refers to both boards when using the term 'Ndigo250M' or 'Ndigo250M-14'. These boards are conceptionally very similar to the Ndigo5G-10 and use an almost identical API. All three types of boards can be mixed and synchronized by using the cronotools software library. The Ndigo5G-10 merges all data channels into a single ring buffer whereas the Ndigo250M has a separate buffer for each channel.

2 Hardware

2.1 Installing the Board

The Ndigo250M-14 board can be installed in any PCIe slot with eight or more lanes. If the slot electrically supports less than 8 lanes, the board will operate at lower data throughput rates.

Using a single Ndigo250M, no further connections need to be made. For applications that require more than four ADC channels, several Ndigo boards can be operated in sync. Any board of the Ndigo product line can be synced to other Ndigo boards, allowing, for instance, for a combination of high speed ADCs (Ndigo5G) and slower high resolution ADCs (Ndigo250M-14) or the upcoming Ndigo TDC.

The signals used for board synchronization and inter-board triggering are transferred on a bus between the boards. Join all C2 connectors (see Figure 2.3 on page 5) on the boards using a ribbon cable. Both ends of the bus need to be terminated properly. If using a Ndigo Crate, connectors providing the termination are located on the crate mainboard next to the PCIe slots to the extreme left and right. For more details, please refer to the Ndigo Crate user guide. In applications that use only a few Ndigo boards installed directly inside a PC, termination PCBs available from cronologic can be used.

Ndigo250M's standard device driver can be used to read out any number of boards and acquire data. For more complex scenarios, using the cronoSync-library, which is part of cronoTools, is recommended. The cronoSync library is provided with the Ndigo device driver. Please refer to the cronoTools user guide for more information.

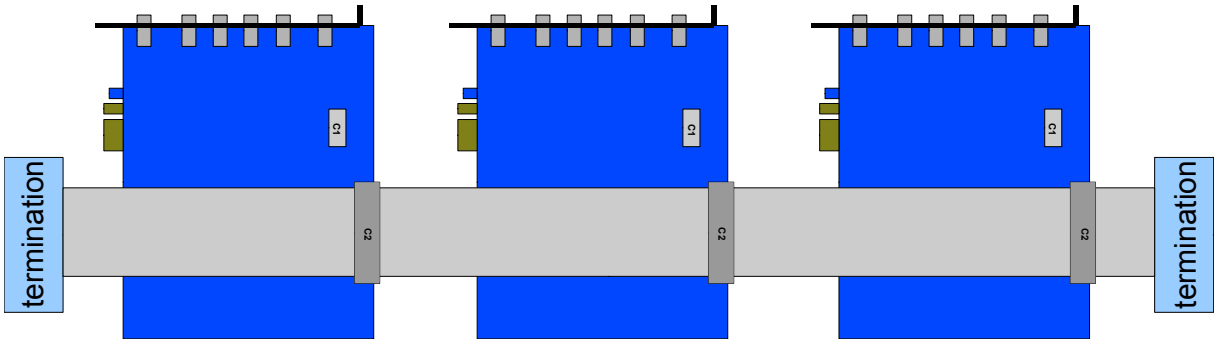

Figure 2.1: If several Ndigo boards are connected to work in sync , the boards must be connected using a ribbon cable as bus for synchronization and trigger signals. Proper termination is required at both ends of the cable.

2.2 External Inputs and Connectors

2.2.1 Connectors

The inputs of the Ndigo250M-14 are located on the PCI bracket. Figure 2.3 on page 5 shows the location of the 4 analog inputs A to D and the two digital inputs G (GATE) and T (TRIGGER). Furthermore, a board interconnection connector (C2) can be found at the top edge of the Ndigo250M-14, as displayed in Figure 2.3 on page 5. Connector C2 is used as a bus interface between multiple Ndigo boards. It is used for distributing clock, trigger and sync signals. Proper termination must be placed at both ends of the bus interconnection ribbon cable, as shown in figure 2.1.

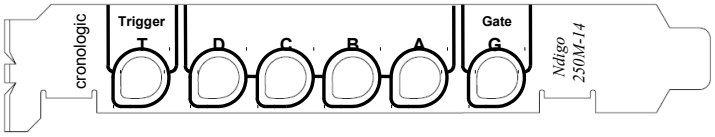

Figure 2.2: Input connectors of an Ndigo250M-14 located on the PCI bracket.

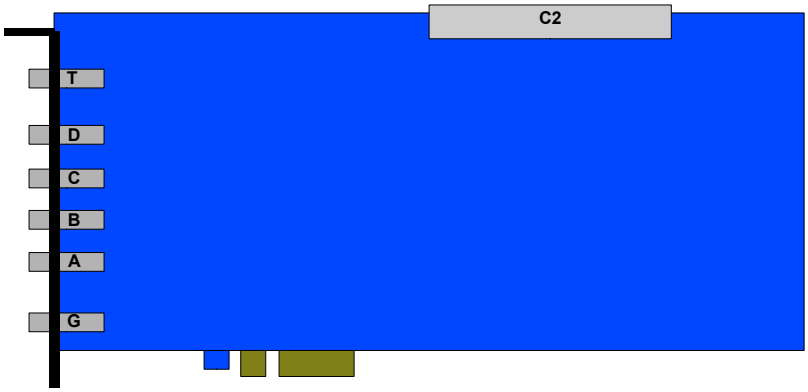

Figure 2.3: Schematics of an Ndigo250M-14 board showing inter-board connector C2.

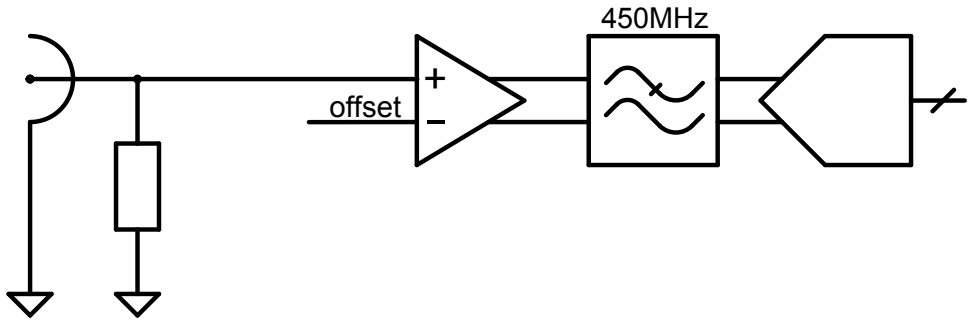

Figure 2.4: Input circuit for each of the four analog channels.

2.2.2 Analog Inputs

The analog inputs of the ADC are single ended LEMO00 coax connectors. The inputs have a 50Ω impedance and are DC coupled. The inputs are converted to a differential signal using an operational amplifier. The input voltage range is $1V_{p\text{-}p}$ centered around a programmable offset voltage. See section 6.4.3 for details.

Analog Offsets

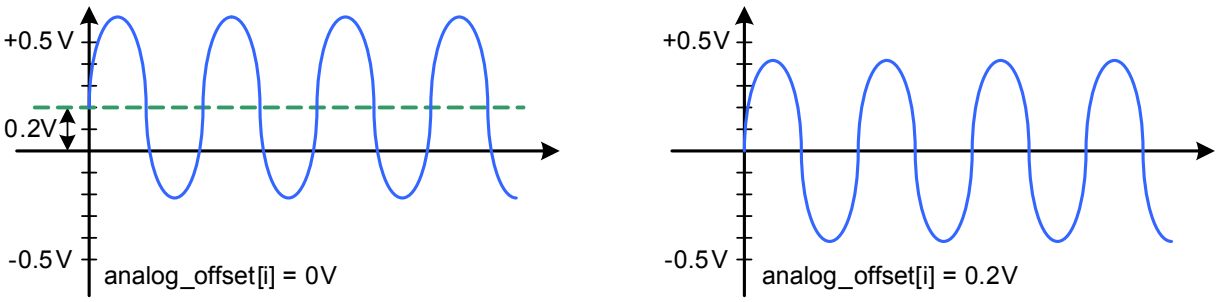

Figure 2.5: Users can subtract an analog offset from the input before sampling at the ADC

Users can set the approximate center of the input voltage range individually for each channel by using the analog_offset parameter. The valid voltage range for the offset is $0 \ldots 2.5V$. All

input parameters are specified in section 6.4.3. This feature is especially useful for highly

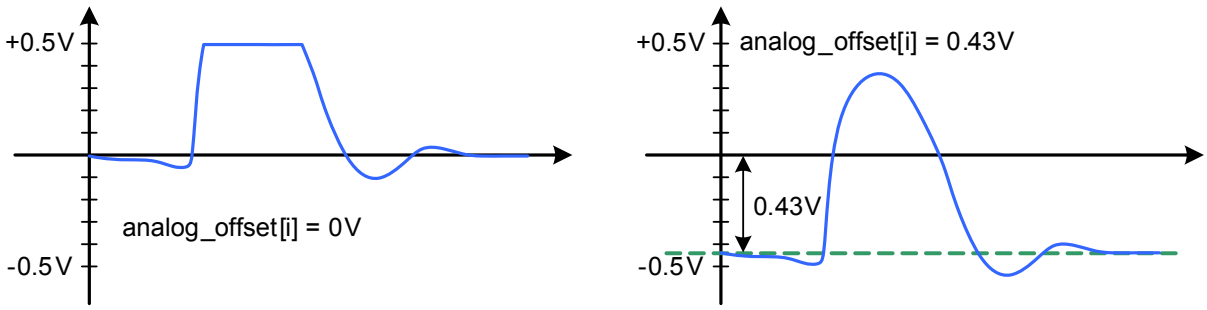

Figure 2.6: Asymmetric signal shifted to increase dynamic range

asymmetric signals, such as pulses from TOF spectrometers or LIDAR systems. By shifting the DC baseline to one end of the ADC range, the input range can be used fully, providing the maximum dynamic range.

2.2.3 Digital Inputs

There are two digital inputs on the front slot cover called TRIGGER and GATE.

Both inputs provide a digital input signal routed to the trigger matrix. These signals can be used to trigger any of the trigger state machines and gating blocks. The inputs are AC coupled. The DC offset is configurable via the parameters analog_offset[4] for TRIGGER and analog_offset[5] for GATE in the configuration structure. See section 3.4.1 for a definition of the configuration structure.

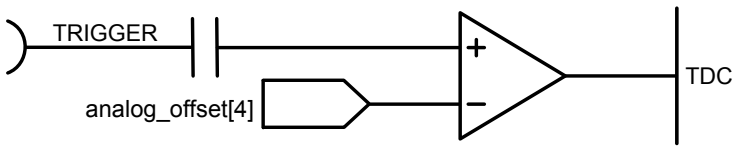

Figure 2.7: TRIGGER and GATE are AC coupled inputs with programmable threshold. (Only TRIGGER shown)

For the digital signals the offset directly sets the discrimination threshold at which edges are detected. To support both negative and positive pulses the valid range is -1.25V to +1.25V. To prevent inter symbol interference caused by the AC coupling, the distance between adjacent pulses should be much longer than the width of each of the pulses.

The FPGA input logic is shown in Figure 2.13 on page 12.

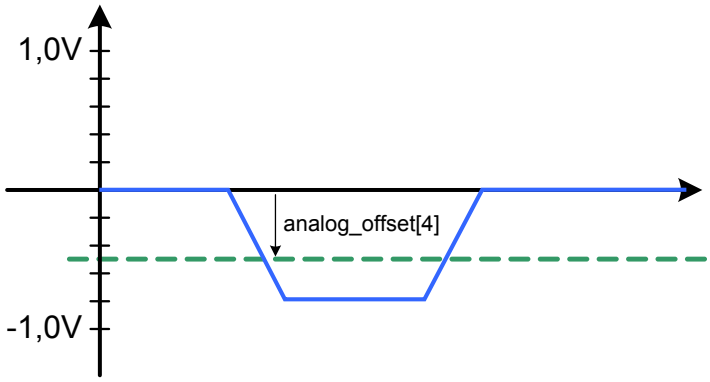

Figure 2.8: For TRIGGER and GATE negative thresholds can be set to support negative signals such as 16mA NIM.

TDC on Input TRIGGER

There is a TDC connected to the input TRIGGER. When used with the TDC, input TRIGGER supports negative pulses only and analog_offset[4] must be negative. The TDC creates packets of type 8. These packets first contain a coarse timestamp and a payload that can be used to calculate the trigger position with higher precision. The function ndigo_process_tdc_packet() can be used to replace the coarse timestamp with the precise timestamp. This function is described in section 3.5 on page 39. TDC pulses must have a minimum duration of 3.3ns. The dead-time of the TDC is 32ns. The TDC is only enabled if the divisor is set to 2. For the Ndigo 125M board the divisor has to be at least 4. Thus the TDC is disabled for this board. For lower sampling frequencies the decimation option can be used.

2.3 Extension Card

This Option is only available for the Ndigo5G-10.

2.4 Ndigo250M-14 Functionality

The analog input signal is quantized to 14 bits. However, the board always scales and offsets the data to 16 bit signed data centered around 0.

Interleaving of multiple channels is not supported

2.4.1 Zero Suppression

One of Ndigo250M's key features is on-board zero suppression to reduce PCIe bus load. Only data that passes specifications predefined by the user is transmitted. This guide refers to the data structure in which wave form data is transmitted as "packets". A packet contains the wave form data and a timestamp giving the absolute time (i.e. the time since start of data acquisition) of the packet's last sample.

Figure 2.9 shows a simple example: Data is written to the PC only if values exceed a specified threshold. Expanding on that, Ndigo250M's zero suppression can be used to realize much more complex scenarios.

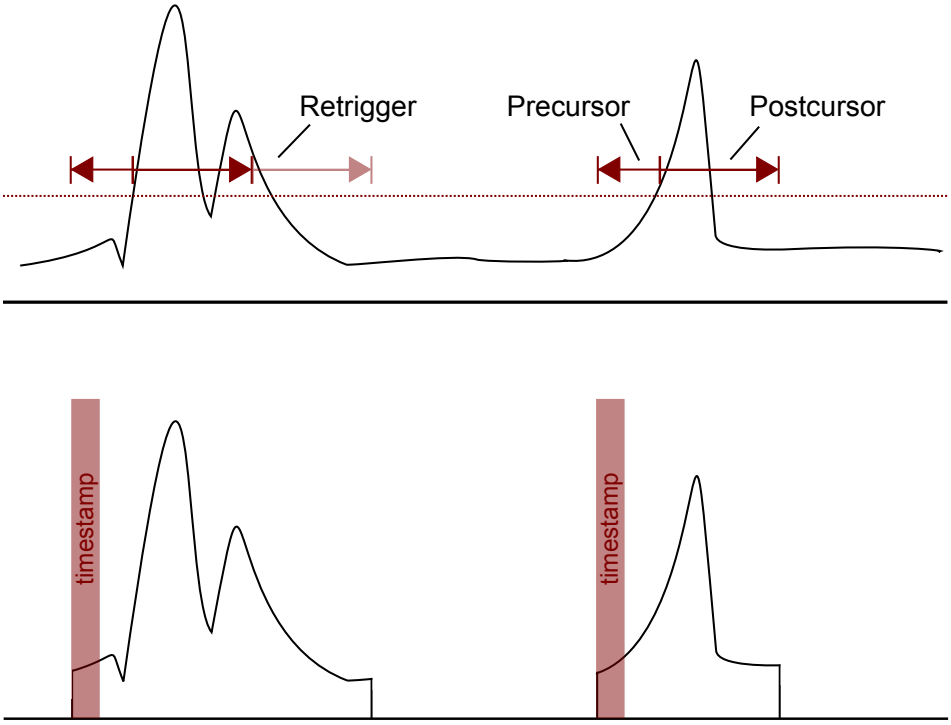

Figure 2.9: Simple zero suppression: Only data with values above a threshold are written to the PC.

2.4.2 Trigger Blocks

The Ndigo250M-14 records analog wave forms using zero suppression. Whenever a relevant waveform is detected, data is written to an internal FIFO memory. Each ADC channel has one trigger block determining whether data is written to the FIFO. The parameters are set in Structure ndigo_trigger_block(See chapter 3.4.3 on page 35).

Each trigger block consists of two independent units that check the incoming raw data stream for trigger conditions (Fig. 2.9 on page 9). Users can specify a threshold and can choose whether triggering happens whenever incoming data is below or above the threshold (level triggering) or only in the moment data crosses the threshold (edge triggering).

A gate length can be set to extend the trigger window. Furthermore, if users choose precursor values > 0, the trigger unit will start writing data to the FIFO precursor samples before the trigger event.

When using edge triggering, all packets have the same length (Figure 2.10 on page 11): precursor + length + 1 cycles. For level triggering, the packet length is data dependent (Figure 2.11 on page 11).

For the Ndigo250M-14 the triggering is sample accurate.

If retriggering is active, the current trigger window is extended if a trigger event is detected inside the window. The extension lasts length cycles after the last sample trigger event.

A trigger block can use several input sources:

- the 8 trigger decision units of all four ADC channels (Figure 2.12 on page 12)
- the GATE input (Figure 2.13 on page 12)
- the TRIGGER input (Figure 2.13 on page 12)
- a function trigger providing random or periodic triggering (Section 2.4.4 on page 16)
- triggers originating from other cards connected with the sync cable or from the Ndigo Extension card (BUS0, BUS1, BUS2, BUS3)
- A second set of trigger units for the digital inputs TRIGGER, GATE, BUS0, BUS1, BUS2, and BUS3 that is set in hardware to positive edge triggering. This set of triggers is not available as inputs for the gate blocks.

Trigger inputs from the above sources can be combined using logical “OR” (Figure 2.14 on page 13) by setting the appropriate bits in the trigger blocks source mask.

Triggers can be fed into the gating blocks described on page 14 (Figure 2.15). Gating blocks can be used to block writing data to the FIFO. That way, only zero suppressed data occurring when the selected gate is active is transmitted. This procedure reduces PCIe bus load even further (Figure 2.15).

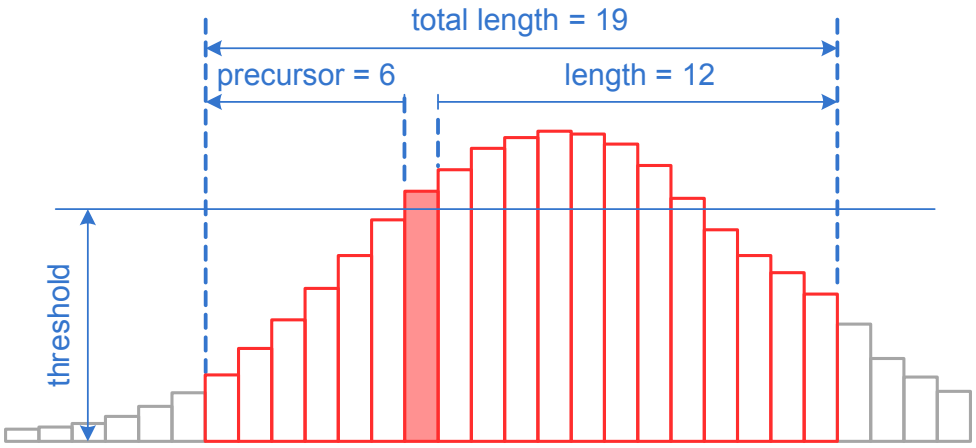

Figure 2.10: Parameters for edge triggering

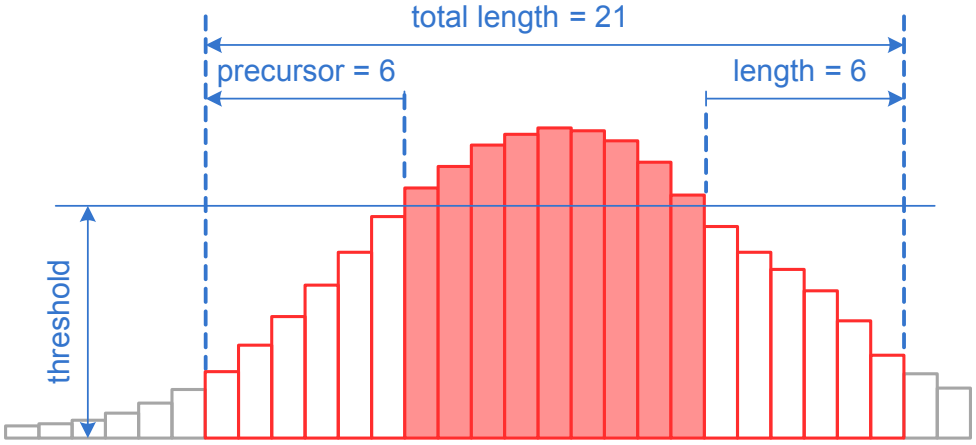

Figure 2.11: Parameters for level triggering

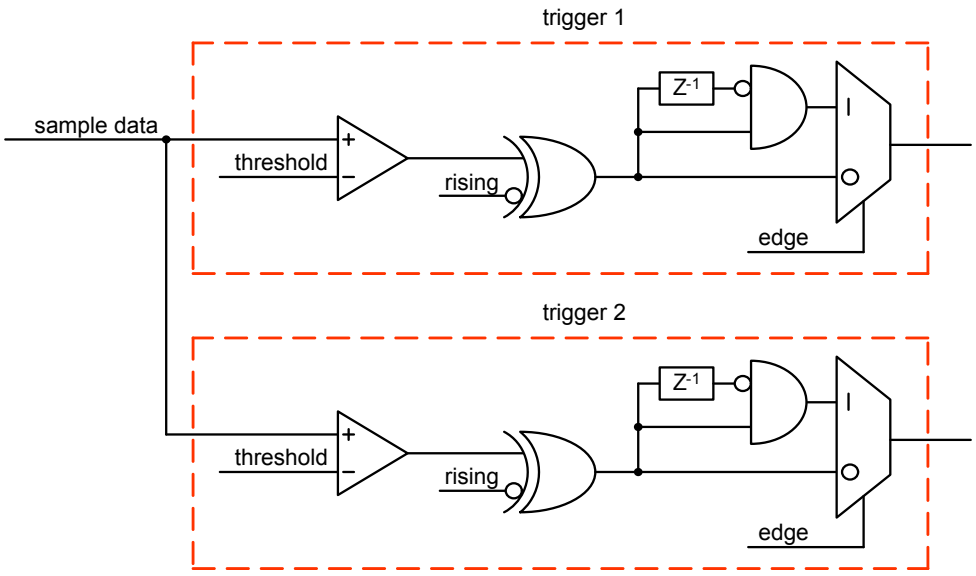

Figure 2.12: From the ADC inputs, a trigger unit creates an input flag for the trigger matrix. Each digitizer channel (A, B, C, D) has two trigger units.

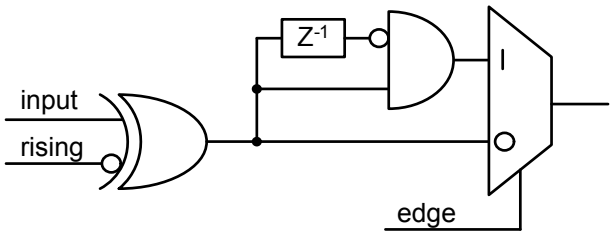

Figure 2.13: The digital inputs TRIGGER, GATE, BUS0, BUS1, BUS2 and BUS3 have trigger units without a programmable threshold.

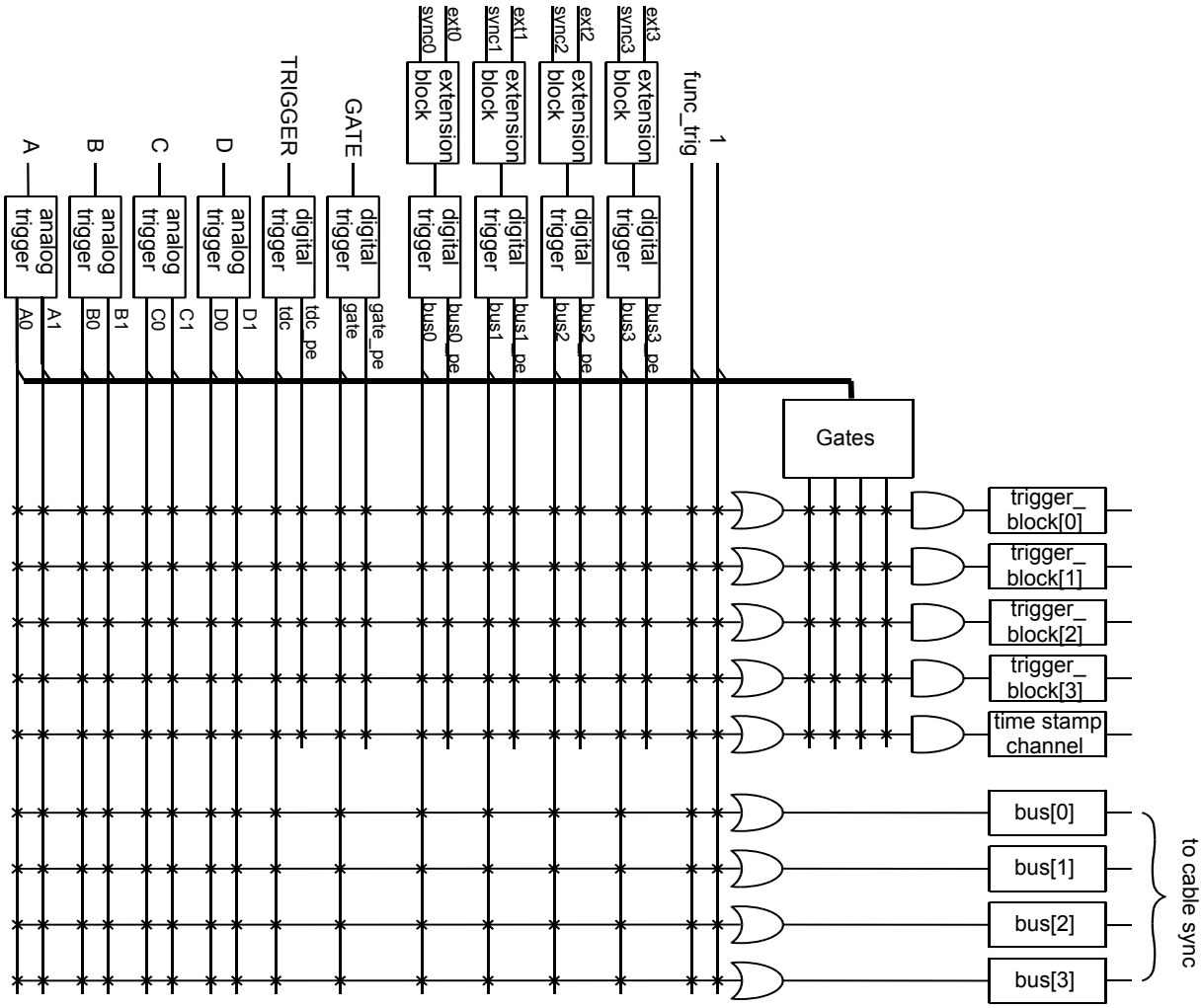

Figure 2.14: Trigger Matrix: The trigger signals of each ADC channel, the TRIGGER input, the GATE input or the sync cable can be combined to create a trigger input for each trigger block. The four gate signals can be used to suppress triggers during certain time frames.

2.4.3 Gating Blocks

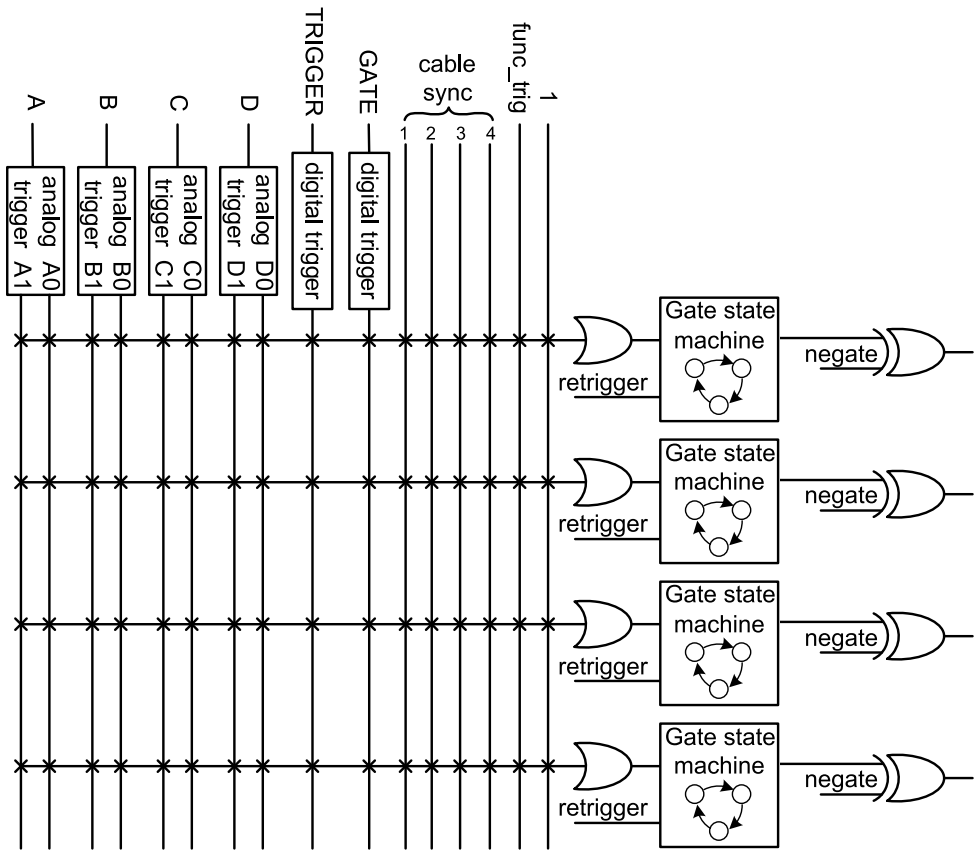

Figure 2.15: Gating Blocks: Each gating block can use an arbitrary combination of inputs to trigger its state machine. The outputs can be individually inverted and routed to the AND-gate feeding the trigger blocks.

To decrease the amount of data transmitted to the PC, the Ndigo250M-14 includes 4 independent gate and delay units. A gate and delay unit creates a gate window starting at a specified time after a trigger, closing the window at gate stop.

Trigger blocks can use the gate signal to suppress data acquisition: Only data that fulfills zero sup-pression specifications occurring in an active gate window is written to the PC.

All triggers from the 4 trigger blocks, the GATE and TRIGGER inputs, triggers from a connected board and the function generator can be used as inputs.

The retrigger feature will create a new gate if a trigger occurs during an active gate window. The gate signal can be inverted, causing an active gate to close for a time defined by the user.

The parameters of a gating block are set in structure ndigo_gating_block described on page 36.

Figure 2.16 shows the functionality of the gate timing and delay unit. Active gate time is marked in green.

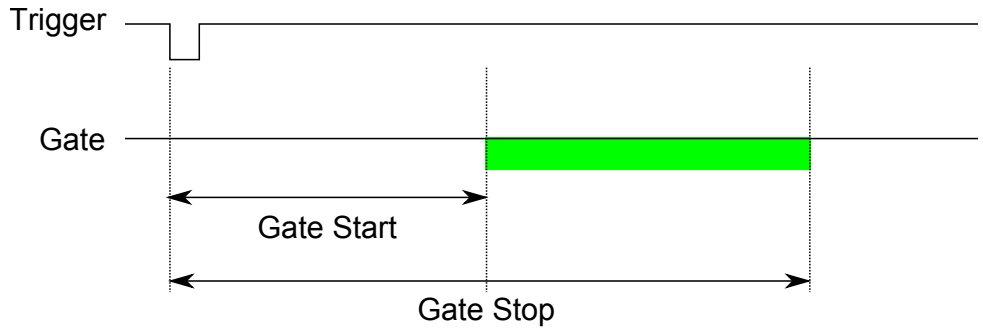

Figure 2.16: Gate and delay functionality: When a trigger occurs, the gate opens after a set period of time ("gate start") and closes when it reaches "gate stop".

Gating Example 1: Suppression of Noise After Starting an Acquisition

In mass spectrometer and other experiments, noise while starting data acquisition can result in undesired trigger events for that time period. To prevent noise in the output data, a gating block could be used to suppress all triggers during start-up.

The following example illustrates the use of a gating block to prevent noise: The GATE input transmits a pulse on each acquisition start. The trigger structure of the GATE input is used to select pulse polarity. Then, the GATE trigger is selected as gating block input and the gating block's start parameter is set to 0. The stop parameter is set to the desired length measured in 3.2ns clock cycle and negate is set to true. The gating block will now output a low pulse of the desired length whenever there is a pulse on the GATE input.

Enabling this gating block as an AND input to the trigger block, for which noise shall be suppressed.

Gating Example 2: Delayed Trigger

To sample a short window at a specified time after a trigger event on a channel, the gating block can be used to create a delayed trigger. To do this, one of the triggers of the channel of interested is configured to the desired parameters by selecting the threshold, setting the edge polarity and enabling edge triggering.

Instead of directly using this trigger as input to the trigger block's input matrix, the trigger is selected as an input to a gating block. The block is configured to $start = delay$ and $stop = start + 1$, $negate = false$. This causes the gating block to produce a one clock cycle pulse on

its output after the specified delay.

To send this pulse to the trigger block, the gating block must be enabled in the trigger block's AND matrix and the ONE trigger source must be selected.

2.4.4 Auto Triggering Function Generator

Some applications require periodic or random triggering. Ndigo250M's function generator provides this functionality.

The delay between two trigger pulses of this trigger generator is the sum of two components: A fixed value M and a pseudo random value given by the exponent N.

The period is

$$T = 1 + M + [1...2^N] \tag{2.1}$$

samples.

This allows to monitor input signals at times the current trigger configuration does not trigger, e. g. to get base line information in mass spectrometry applications. It can also be used to determine a suitable threshold level for the trigger by first getting random statistics on the input signal.

Auto Triggering Example 2: Continuous Data Retrieval

With the Ndigo250M it is possible to sample and process data continuously. For that, one defines the length of the packet and configures the auto trigger accordingly.

Choosing the length of a packet is always a tradeoff between having to process headers for short packets (see further down and 4.0.2) and increasing the latency for long packets. The latter is due to the fact, that the driver only returns complete packets.

The length of the data is set in words (maximum 4093) which contain 4 samples of length 4ns each. The packet thus includes a header (see 4.0.2), the timestamp of the last word in the packet and the data. Using pointer arithmetics one can directly access the data in the packet, ignoring header and timestamp.

Setting the auto trigger analogously to equation 2.1 to $M = S - 1$ and $N = 0$, with S the number of samples in each packet, one obtains a closed cycle of triggering and data recording. To enable the auto trigger one has to configure the trigger block of the current channel with *length* $= S$ and *sources* $=$ *NDIGO_TRIGGER_SOURCE_AUTO* and enable it.

2.4.5 Timestamp Channel

The timestamp channel produces a stream of small packets that denote the time of the trigger event. An arbitrary set of trigger sources can be selected in the trigger matrix to cause the creation of a packet.

The packets have a fixed length of 16 bytes. The format is described on page 43. The length field of the packet contains a 32 bit pattern that contains the levels of all trigger sources at the time of the trigger event except for the period monitor. Only one packet is created, no matter how many trigger sources caused the timestamp channel to trigger. The timestamp channel can be gated.

2.4.6 Data Lookup Table

This feature ist not available for the Ndigo250M-14.

2.5 Multiple Ndigo boards synchronization

Using several Ndigo devices in applications that use more channels than a single board can provide requires synchronized operation. To ensure exact synchronization, a delay parameter needs to be set for each board. This parameter might change in case boards are swapped, added or removed and in some cases might change after a firmware update.

The calibration tool "MultiboardCalibration.exe" is available after installing the Ndigo device driver. It is used to find appropriate delay values for each board in a given board setup. After starting, the application lists all Ndigo boards found (Figure 2.17).

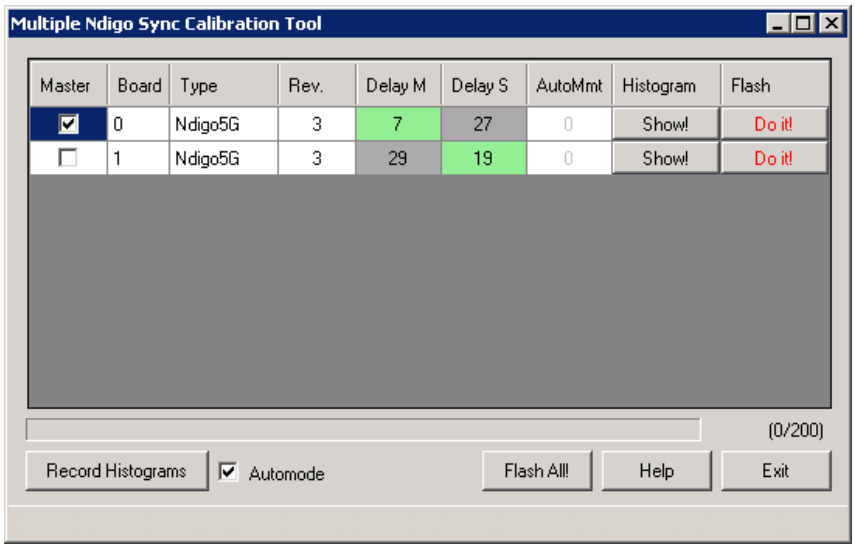

Figure 2.17: Main window of the multiple boards sync calibration tool.

A board's appropriate delay depends on whether it operates in master or slave mode. The respective values can be set in the column "Delay M" (for master boards) and "Delay S" (for slave boards). The designated master board can be selected in the column "Master". The calibration procedure creates a histogram for each board, displaying the current delay between the boards. The histogram can be viewed by clicking on "Show!". When the appropriate delay values are found, they can be stored in the on-board flash prom by clicking "Do it!" separately for each board. Clicking "Flash All!" will write the values to all boards at once. Please note: Flashing the values might take up to 10 seconds during which the program might not respond.

Important note:If the application reports a "PLL not locked" error check the cable. If the recording of histograms does not make progress check the cable. Make sure the cable is properly terminated at both ends and firmly attached to each card.

2.5.1 Calibration Procedure

1. Make sure the “Automode” is selected.
2. Record the calibration histograms by pressing “Record histograms”. The program will perform up to 200 measurements of the sync delay. After accumulating some data, the delay values found are reported in the column “AutoMmt”. The values can be verified by examining histogram that was recorded. A board’s histogram should look like the one shown in Figure 2.18. During normal operation the delay will be adjusted such, that the data points accumulated roughly coincide with the vertical markers in the upper panel. As the delay pattern is periodic, valid delay values are between 0 and 31. Thus, the delay value found by the auto measurement should correspond to the distance between the vertical markers and accumulated data points. Hint: when moving the mouse pointer across the histogram the delay value of the current location is displayed.
3. After stopping the data acquisition (by pressing “Record Histograms” again) the delay values of the auto measurement need to be copied to the columns “Delay M” or “Delay S” depending on the corresponding board being a master or a slave. The correct field to copy the value to is highlighted in green.
4. you may record recorded a new dataset as a crosscheck that the delay is now set to an appropriate value. By disabling “Automode” the new delay values are used. Press “Record Histograms” in order to start the data acquisition. After some time the histogram should look similar to the one in Figure 2.19.
5. The delay values for all boards in a set needs to be found. For the case a board acts as a master, the value “Delay M” needs to be adjusted, in case it is a slave, the “Delay S” parameter needs to be changed. In order to find the master-case delay values for all boards, the calibration procedure needs to be performed with every board acting as a master once. After changing the master board, the slave values of the other boards don’t need to be readjusted. Only Ndigo5G boards may be set as masters. Therefore, a Ndigo250M board only needs to be calibrated as a slave.
6. After finding all delay values, write the values to the on-board flash PROMs by pressing “Flash All!”.

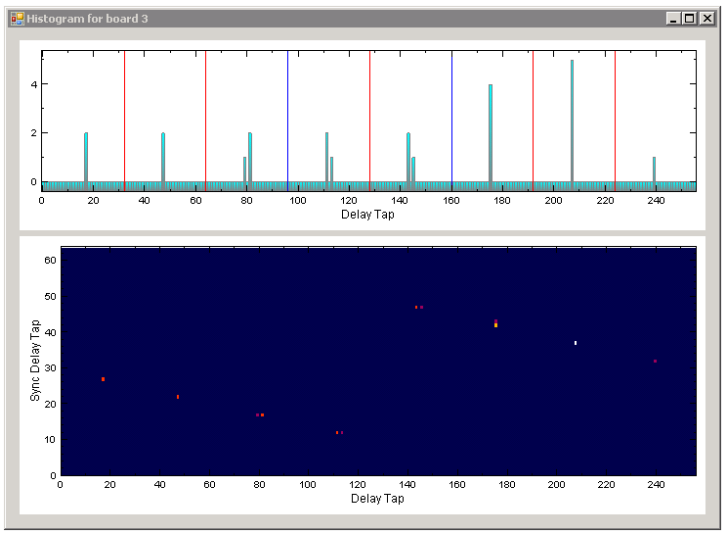

Figure 2.18: Histogram for the case the delay value for the board is not set correctly. Please note: the lower panel might differ from board to board, with the "step" being at a different position.

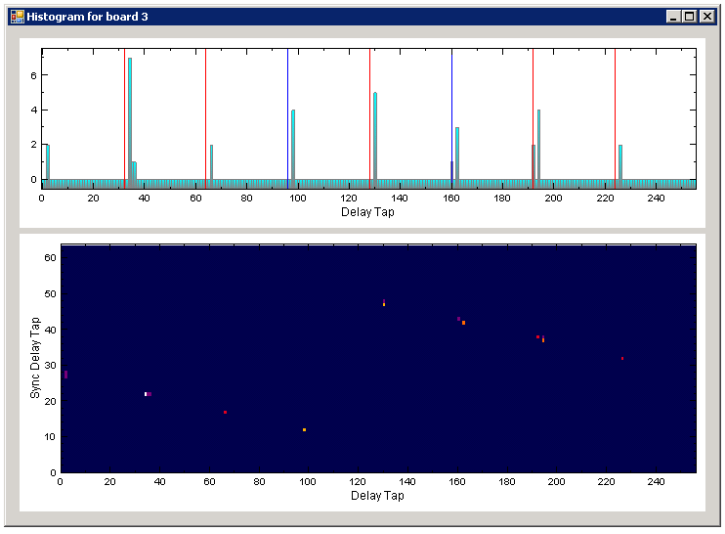

Figure 2.19: Histogram for the case the delay value for the board is set correctly. Please note: the lower panel might differ from board to board, with the "step" being at a different position.

2.5.2 Synchronizing with an HPTDC8-PCI

The Ndigo250M-14 does not support synchronization with an HPTDC8-PCI.

2.6 Performing a firmware update

After installing the Ndigo device driver, a firmware update tool is available. By choosing "NdigoFirmwareGUI.exe" a firmware update can be performed. After invoking the application a window as shown in Figure 2.20 will appear. The tool can be used for updating the firmware and to create a backup of the on-board calibration data of the Ndigo unit. If several boards are present, the one which is going to be used can be selected in the upper left corner of the window.

"Backup" When pressing the "Backup" buttons a backup of the firmware or the calibration data will be created, respectively.

"Browse" In order to perform a firmware update, chose the ".ndigorom"-file to be used by pressing "Browse". The file contains the firmware proms for all boards of the Ndigo product line.

"Flash" By pressing "Flash" the firmware is written to the board.

"Verify" "Verify" can be used to compare the data stored inside the prom to the one inside a file.

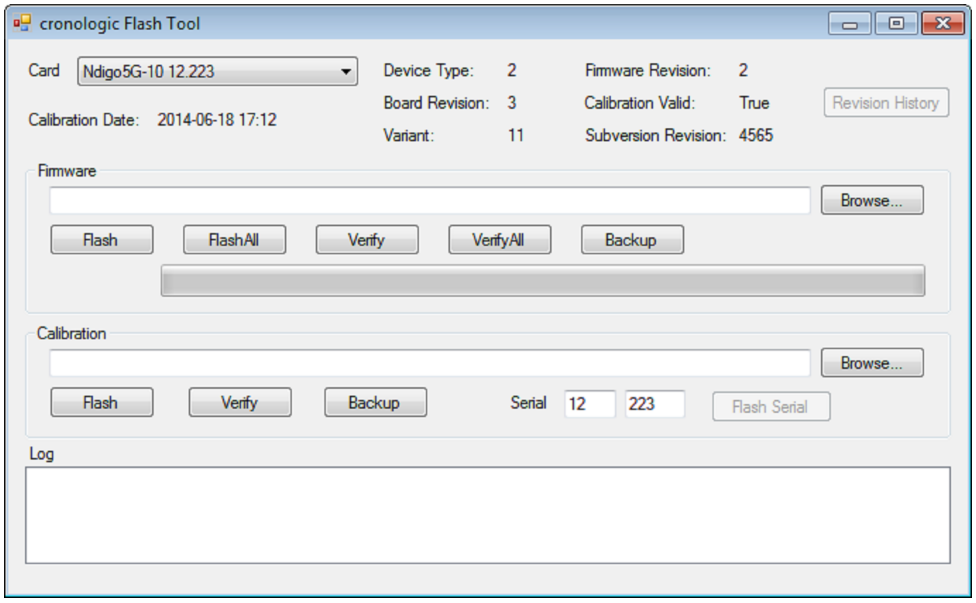

Figure 2.20: The firmware update and calibration data backup tool as provided with the Ndigo device driver.

Important note: The new firmware will only be used after a power cycle, i.e. after switching the PC (or Ndigo crate) off and back on. A simple reboot is not sufficient. Therefore the

information shown in the upper half of the application window does not change right after flashing a new firmware.

2.7 Calibrating the TDC

After each update of the Ndigo250M-14 firmware the TDC has to be calibrated. The calibration is done with the tool "TDC_Calibration.exe" which is available after installing the Ndigo device driver. After invoking the application a window as shown in Figure 2.21 will appear.

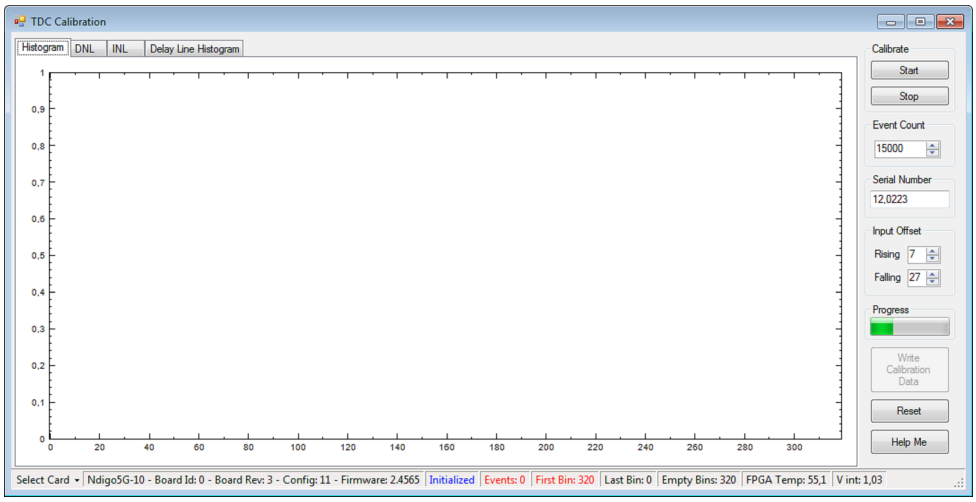

Figure 2.21: The TDC calibration tool as provided with the Ndigo device driver.

The calibration procedure is as follows:

1. Connect an external pulse signal to the Trigger input. The signal should be low active with a frequency in the kHz range. It must not be synchronized to the clock source of the Ndigo250M-14. The input frequency must not exceed 10 MHz. The pulse low and high width has to be at least 10ns each.
2. Set *Serial Number* according to the sticker on the card if the shown value is not correct.
3. Start capturing pulse events by pressing the *Start* button.
4. Adjust the *Input Offset* so that *First Bin* is in the range of 4 to 16. *First Bin* is less than 4, increment *Input Offset* by one. If *First Bin* is greater than 16 decrement *Input Offset* by one. Repeat increment/decrement until *First Bin* is in the range of 4 to 16. Depending on the firmware revision the *Input Offset* value for a successful calibration may be in the range of 6 to 10 or 28 to 32.

5. When the *Write Calibration Data* button becomes enabled press it to update the calibration data on the card.
6. Calibration done!

The card can only be successfully calibrated if:

- *First Bin* is in the range of 4 to 16
- *Empty Bins* is less than (First Bin + 4)
- at least 10,000 events have been captured
- a valid serial number is set.

Important note: If the application reports an error check if the input pulse is within specification.

3 Driver Programming API

The API is a DLL with C linkage. There also exists a .Net wrapper.

The functions provided by the DLL are declared in Ndigo250M_interface.h and Ndigo_common_interface.h.

3.1 Constants

#define NDIGO250M_CHANNEL_COUNT 4
The number of analog input channels.

#define NDIGO250M_GATE_COUNT 4
The number of gating blocks.

#define NDIGO250M_TDC_COUNT 1
The number of TDC channels.

#define NDIGO250M_DMA_COUNT 6
The number of DMA channels. Unlike the Ndigo5G-10, this device has an individual DMA buffer for each data source.

#define NDIGO250M_FIFO_DEPTH 4094
Number of samples that can be buffered before readout. Longer packets are possible under certain circumstances.

3.2 Initialization

int ndigo250m_count_devices(int *error_code, char **error_message)
Return the number of boards that are supported by this driver in the system.

int ndigo250m_get_default_init_parameters(ndigo_init_parameters *init)
Get a set of default parameters to feed into ndigo_init(). This must always be used to initialize the ndigo_init_parameter structure.

ndigo_device *ndigo250m_init(ndigo_init_parameters *params, int *error_code, char **error_message)
Open and initialize the Ndigo250M-14 board with the given index. With error_code and error_message the user must provide pointers where to buffers where error information should be written by the driver. The buffer for the error message must by at least 80 chars long.

Params is a structure of type ndigo_init_parameters that must be completely initialized.

int ndigo_close(ndigo_device *device)
Finalize the driver for this device.

3.2.1 Structure ndigo_init_parameters

int version
Must be set to NDIGO_API_VERSION

int card_index
The index in the list of Ndigo250M-14 boards that should be initialized. There might be multiple boards in the system that are handled by this driver as reported by ndigo_count_devices. This index selects one of them. Boards are enumerated depending on the PCIe slot. The lower the bus number and the lower the slot number the lower the card index.

int board_id
This 8 bit number is filled into each packet created by the board and is useful if data streams of multiple boards will be merged. If only Ndigo250M-14 cards are used this number can be set to the card index. If boards of different types that use a compatible data format are used in a system each board should get a unique id. Can be changed with int ndigo_set_board_id(ndigo_device *device, int board_id).

ndigo_bool_t use_external_clock
Use 10MHz clock supplied by IPC flat band cable. Must be set for all slaves when synchronizing multiple boards.

ndigo_bool_t drive_external_clock
Drive internal 10MHz clock of this board to IPC flat band cable. Must be set for master when synchronizing multiple boards.

ndigo_bool_t is_slave
Data acquisition of this board is controlled by the master board.

int sync_period
Period of the multicard sync pulse. Ignored for single board setups. Should be set to 4 if only Ndigo5G and Ndigo250M boards are present in the system. If other boards are synchronized the correct value is the lowest common denominator of the value of all boards.

int sync_delay
Fine tap delay for incoming sync signals. Ignored for single board setups.

int sync_delay_master
Fine tap delay for sync signals. Ignored for single board setups.

ndigo_bool_t force_window_calibration
If true/1, a valid data window is automatically detected at initialization. Default value is false/0: values from flash memory are used in order to set the data window to correct position.

ndigo_bool_t hptdc_sync_enabled
Must be set to false.

__int64 buffer_size[8]
The minimum size for each of the DMA buffers. The driver will allocate a multiple of the system page size (usually 4096 bytes). The minimum buffer size is four pages. If set to 0 the default size of 16MByte is used. Ndigo250M-14 uses buffer_size[0] to buffer_size[NDIGO250M_DMA_COUNT-1].

int buffer_type
Must be set to NDIGO_BUFFER_ALLOCATE.

__int64 buffer_address
Ignored. Might be used for future buffer types.

int variant
Set to 0. Can be used to activate future device variants such as different base frequencies.

int device_type
Initialized by ndigo_get_default_init_parameters(). Must be left unchanged.

```
#define CRONO_DEVICE_HPTDC 0
#define CRONO_DEVICE_NDIGO5G 1
#define CRONO_DEVICE_NDIGO250M 2
```

int dma_read_delay
Initialized by ndigo_get_default_init_parameters(). The write pointer updated is delay by this number of 4n clock periods to hide race conditions between software and DMA.

3.3 Status Information

3.3.1 Functions for Information Retrieval

The driver provides functions to retrieve detailed information on the type of board, its configuration, settings and state. The information is split according to its scope and the computational requirements to query the information from the board.

int ndigo_get_static_info(ndigo_device *device,ndigo_static_info *info)
This structure contains information about the board that does not change during run time.

int ndigo_get_param_info(ndigo_device *device, ndigo_param_info *info)
The structure returned by this call contains information that changes indirectly due to configuration changes.

int ndigo_get_fast_info(ndigo_device *device, ndigo_fast_info *info)
This call returns a structure that contains dynamic information that can be obtained within a few microseconds.

int ndigo_get_slow_info(ndigo_device *device, ndigo_slow_info *info)
The data reported in this structure requires milliseconds to be obtained. The application should only call it in situation where the program flow can cope with an interruption of that magnitude.

const char* ndigo_get_last_error_message(ndigo_device *device)

3.3.2 Structure ndigo_static_info

This structure contains information about the board that does not change during run time. It is provided by the function ndigo_get_static_info.

int size
The number of bytes occupied by the structure

int version
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

int board_id
Index of the board as passed to the constructor or set via int ndigo_set_board_id(ndigo_device *device, int board_id).

int driver_revision
The lower three bytes contain a triple level hierarchy of version numbers, e.g. 0x010103 encodes version 1.1.3.

A change in the first digit generally requires a recompilation of user applications. Change in the second digit denote significant improvements or changes that don't break compatibility and the third digit changes with minor bugfixes and similar updates.

int firmware_revision
Firmware revision of the FPGA configuration. This increments only when there is a functional change.

int board_revision

int board_configuration
Describes the schematic configuration of the board.

int adc_resolution
Number of bits of the ADC, set to 0 if unknown. Should be 14.

double nominal_sample_rate
Maximum sample rate. 2.5e8 = 250Msps for the Ndigo250M and 1.25e8 = 125Msps for the Ndigo125M.

double analog_bandwidth
1.25e8 for 125MHz.

int chip_id
16 bit factory ID of the ADC chip

int board_serial
Serial number with the year minus 2000 in the highest 8 bits of the integer and a running number in the lower 24 bits. This number is identical with the one on the label on the board.

int flash_serial_low
int flash_serial_high
64 bit manufacturer serial number of the flash chip.

int flash_valid
If not 0 the driver found valid calibration data in the flash on the board and is using it.

ndigo_bool_t dc_coupled
Returns true for the Ndigo250M.

int subversion_revision
A number to track builds of the firmware in more detail than the firmware revision. It changes with every change in the firmware, even if there is no visible effect for the user.

char calibration_date[20]
DIN EN ISO 8601 string YYYY-MM-DD HH:DD describing the time when the card was calibrated.

3.3.3 Structure ndigo_param_info

int size
The number of bytes occupied by the structure.

int version
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

double bandwidth
Analog Bandwidth setting of the ADC. Should be 450MHz.

double sample_rate
Sample rate currenly used. sample_rate = $2.5e8$/divisor.

double sample_period
The period one sample in the data represents in picoseconds.

int board_id
The number the board uses to identify the data source in the output data stream.

int channels
Number of channels. Should be 4.

int channel_mask
Mask with a set bit for each enabled input channel.

__int64 total_buffer
The total amount of the DMA buffer in bytes.

3.3.4 Structure ndigo_fast_info

This structure contains information that can be retrieved within microseconds.

int size
The number of bytes occupied by the structure

int version
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

int adc_rpm
Always reports 0.

int fpga_rpm
Speed of the FPGA fan. Reports 0 if no fan is present.

int alerts

Alert flags from the system monitor and temperature sensor. Bits 0 to 3 are measured by the system monitor of the FPGA. Bits 4 and 5 are provided by an external temperature sensor.

Bit 0 : FPGA temperature alert ($> 85°C$)
Bit 1 : Internal FPGA voltage out of range ($< 1.01V$ or $> 1.08V$)
Bit 2 : FPGA auxiliary voltage out of range. ($< 2.375V$ or $> 2.625V$)
Bit 3 : FPGA temperature critical ($> 125°C$)
Bit 4 : FPGA temperature alert ($> 90°C$)
Bit 5 : FPGA temperature critical ($> 100°C$)

double voltage_aux
Auxiliary FPGA voltage, nominal 2.5V

double voltage_int
Internal FPGA voltage, nominal 1.03V

double fpga_temperature
In °C measured on die by internal system monitor.

double pcie_pwr_mgmt
Set to 0 if link power management is turned off.

int pcie_link_width
Number of PCIe lanes that the card uses. Should be 8. Lower values possible if the mainboard does not support 8 lanes in the slot chosen.

int pcie_max_payload
Maximum size in bytes for one PCIe transaction, depends on system configuration.

3.3.5 Structure ndigo_slow_info

This structure contains data that requires many milliseconds to retrieve.

int size
The number of bytes occupied by the structure.

int version
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar. Set to 0 for all versions up to first release.

double FPGA_temperature
ADC temperature in $°C$ measured on die by external temperature sensor.

double board_temperature
In $°C$. Measured by external temperature sensor.

3.4 Configuration

The device is configured with a configuration structure. The user should first obtain a structure that contains the default settings of the device read from an on board ROM , than modify the structure as needed for the user application and use the result to configure the device.

int ndigo250m_get_default_configuration(ndigo_device *device, ndigo250m_configuration *config)

int ndigo250m_get_current_configuration(ndigo_device *device, ndigo250m_configuration *config)

int ndigo250m_configure(ndigo_device *device, ndigo250_configuration *config)

int ndigo_set_board_id(ndigo_device *device, int board_id)
The board_id can be changed after initialization of the card. If cronotools are used the board_id changes have to be done before cronotools initialization.

3.4.1 Structure ndigo250m_configuration

This is the structure containing the configuration information. It is used in conjunction with ndigo_get_default_configuration, ndigo_get_current_configuration and ndigo_configure.

It uses internally the structures ndigo_trigger_block and ndigo_trigger.

int size
The number of bytes occupied by the structure.

int version
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar. Set to 0 for all versions up to first release.

int reserved1
Reserved for internal usage. Do not change.

ndigo_bool_t tdc_enabled
Enable capturing of TDC measurements on external digital input channel. Only possible for **divisor** = 2 for the Ndigo 250M-14 and not possible at all for the Ndigo 125M. For lower sampling rates the decimation option can be used.

ndigo_bool_t tdc_fb_enabled
Enable enhanced TDC resolution. Currently not implemented.

double analog_offset[6]
Sets the input DC offset-values to +- this value in volts. Valid range for analog_offset[0] to analog_offset[3] is 0 to 2.5V (see section 2.2.2). analog_offset[4] and analog_offset[5] are the

TRIGGER and GATE input thresholds.

ndigo_trigger trigger[NDIGO_TRIGGER_COUNT]
Configuration of the external trigger sources. Threshold is ignored for entries 8 and above.

The trigger indexes refer to the entry in the trigger array and are defined like this:

```
#define NDIGO_TRIGGER_A0          0
#define NDIGO_TRIGGER_A1          1
#define NDIGO_TRIGGER_B0          2
#define NDIGO_TRIGGER_B1          3
#define NDIGO_TRIGGER_C0          4
#define NDIGO_TRIGGER_C1          5
#define NDIGO_TRIGGER_D0          6
#define NDIGO_TRIGGER_D1          7
#define NDIGO_TRIGGER_TDC         8
#define NDIGO_TRIGGER_GATE        9
#define NDIGO_TRIGGER_BUS0        10
#define NDIGO_TRIGGER_BUS1        11
#define NDIGO_TRIGGER_BUS2        12
#define NDIGO_TRIGGER_BUS3        13
#define NDIGO_TRIGGER_AUTO        14
#define NDIGO_TRIGGER_ONE         15
#define NDIGO_TRIGGER_TDC_PE      16
#define NDIGO_TRIGGER_GATE_PE     17
#define NDIGO_TRIGGER_BUS0_PE     18
#define NDIGO_TRIGGER_BUS1_PE     19
#define NDIGO_TRIGGER_BUS2_PE     20
#define NDIGO_TRIGGER_BUS3_PE     21
```

Sources 16 to 21 always are positive edge triggered.

ndigo_trigger_block trigger_block[NDIGO_CHANNEL_COUNT + 1]
A structure describing the trigger settings of the four channels plus the timestamp channel.

ndigo_gating_block gating_block[NDIGO250M_GATE_COUNT]
A structure describing the gating blocks that can be used by the trigger blocks to filter triggers.

int drive_bus[4]
Enable output drive for each of the four external sync lines. Each integer represents a bitmask selecting the trigger sources for that line. The bit mapping is described in section "Structure ndigo_trigger_block" on page 35.

int auto_trigger_period
int auto_trigger_random_exponent
Create a trigger either periodically or randomly. There are two parameters M = trigger_period

and N = random_exponent that result in a distance between triggers of

$$T = 1 + M + [1...2^N] \tag{3.1}$$

clock cycles.

$$0 \leq M < 2^{32} \tag{3.2}$$
$$0 \leq N < 32 \tag{3.3}$$

There is no enable or reset as the usage of this trigger can be configured in the trigger block channel source field.

int divisor
The 500MHz base clock is divided by this divisor to create the rsampling frequency. Valid values are 2 to 32 for the Ndigo250M and 4 to 32 for the Ndigo125M.

int decimation
Reduces sample rate by given factor.
Range: 1 - 255
Note: Decimation factors 2 and 3 only skip samples, factors 4 - 255 use a CIC decimation filter.

int high_gain[2]
If set to 1 the input gain is boosted by 3.5dB. Gain[0] modifies channels 0 and 1, gain[1] modifies channels 2 and 3. This feature is not yet implemented.

int fine_gain[2]
Gain setting in steps of 0.5dB from 0 (0dB) to 12 (6.0dB). Gain[0] modifies channels 0 and 1, gain[1] modifies channels 2 and 3. This feature is not yet implemented.

int gain_correction[2]
Gain setting in steps of 0.05dB from 0 (0dB) to 10 (0.5dB). Gain[0] modifies channels 0 and 1, gain[1] modifies channels 2 and 3. This feature is not yet implemented.

3.4.2 Structure ndigo_trigger

short threshold
Sets the threshold for the trigger block within the range of the ADC data of -32768 and +32768.

For trigger indices NDIGO_TRIGGER_TDC to NDIGO_TRIGGER_BUS3_PE the threshold is ignored.

ndigo_bool_t edge
If set this trigger implements edge trigger functionality else this is a level trigger.

For trigger indices NDIGO_TRIGGER_AUTO and NDIGO_TRIGGER_ONE this is ignored.

For trigger indices NDIGO_TRIGGER_TDC_PE to NDIGO_TRIGGER_BUS3_PE this must be set.

ndigo_bool_t rising
If set trigger on rising edges or when above threshold.

For trigger indices NDIGO_TRIGGER_AUTO and NDIGO_TRIGGER_ONE this is ignored.

For trigger indices NDIGO_TRIGGER_TDC_PE to NDIGO_TRIGGER_BUS3_PE this must be set.

3.4.3 Structure ndigo_trigger_block

ndigo_bool_t enabled
Activate triggers on this channel.

ndigo_bool_t retrigger
If a new trigger condition occurs while the postcursor is acquired the packet is extended by starting a new postcursor. Otherwise the new trigger is ignored and the packet ends after the precursor of the first trigger.

The retrigger setting is ignored for the timestamp channel.

ndigo_bool_t reserved1
Defaults to false. Do not change.

ndigo_bool_t reserved2
Defaults to false. Do not change.

int precursor
The amount of data preceding a trigger that is captured.

The precursor setting is ignored for the timestamp channel.

int length
The total amount of samples that are recorded in addition to the trigger window. Precursor determines how many of these are ahead of the trigger and how many are appended after the trigger. In edge trigger mode the trigger window always is one sample, in level trigger mode it is as long as the trigger condition is fulfilled.

The length setting is ignored for the timestamp channel.

int sources
A bit mask with a bit set for all trigger sources that can trigger this channel.

```
#define NDIGO_TRIGGER_SOURCE_A0          0x00000001
#define NDIGO_TRIGGER_SOURCE_A1          0x00000002
#define NDIGO_TRIGGER_SOURCE_B0          0x00000004
```

```
#define NDIGO_TRIGGER_SOURCE_B1          0x00000008
#define NDIGO_TRIGGER_SOURCE_C0          0x00000010
#define NDIGO_TRIGGER_SOURCE_C1          0x00000020
#define NDIGO_TRIGGER_SOURCE_D0          0x00000040
#define NDIGO_TRIGGER_SOURCE_D1          0x00000080
#define NDIGO_TRIGGER_SOURCE_TDC         0x00000100
#define NDIGO_TRIGGER_SOURCE_GATE        0x00000200
#define NDIGO_TRIGGER_SOURCE_BUS0        0x00000400
#define NDIGO_TRIGGER_SOURCE_BUS1        0x00000800
#define NDIGO_TRIGGER_SOURCE_BUS2        0x00001000
#define NDIGO_TRIGGER_SOURCE_BUS3        0x00002000
#define NDIGO_TRIGGER_SOURCE_AUTO        0x00004000
#define NDIGO_TRIGGER_SOURCE_ONE         0x00008000
#define NDIGO_TRIGGER_SOURCE_TDC_PE      0x01000000
#define NDIGO_TRIGGER_SOURCE_GATE_PE     0x02000000
#define NDIGO_TRIGGER_SOURCE_BUS0_PE     0x04000000
#define NDIGO_TRIGGER_SOURCE_BUS1_PE     0x08000000
#define NDIGO_TRIGGER_SOURCE_BUS2_PE     0x10000000
#define NDIGO_TRIGGER_SOURCE_BUS3_PE     0x20000000
```

int gates

```
#define NDIGO_TRIGGER_GATE_NONE   0x0000
#define NDIGO_TRIGGER_GATE_0      0x0001
#define NDIGO_TRIGGER_GATE_1      0x0002
#define NDIGO_TRIGGER_GATE_2      0x0004
#define NDIGO_TRIGGER_GATE_3      0x0008
```

double minimum_free_packets;
This parameter sets how many packets are supposed to fit into the on-board FIFO before a new packet is started after the FIFO was full, i.e. a certain amount of free space in the FIFO is demanded before a new packet is written after the FIFO was full. As a measure for the packet length the gatelength set by the user is used. The on-board algorithm checks the free FIFO space only in case the FIFO is full. Therefore, if this number is 1.0 or more at least every second packet in the DMA buffer is guaranteed to have the full length set by the gatelength parameters. In many cases smaller values will also result in full length packets. But below a certain value multiple packets that are cut off at the end will show up.

3.4.4 Structure ndigo_gating_block

ndigo_bool_t negate
Invert output polarity. Defaults to false.

ndigo_bool_t retrigger
Defaults to false. If retriggering is enabled the timer is reset to the value of the start parameter whenever the input signal is set while waiting to reach the stop time.

ndigo_bool_t extend
Defaults to true. If set, a gate is created with the set timing from the first occurrence of the input trigger even for short gates. If not set, the input signal must persist for the gate to be created. This feature is NOT YET IMPLEMENTED.

ndigo_bool_t reserved1
Defaults to false. Do not change.

int start
The number of samples from the first input signal seen in the idle state until the gating output is set.

int stop
The number of samples from leaving the idle state until the gating output is reset. If retriggering is enabled the timer is reset to the value of the start parameter whenever the input signal is set while waiting to reach the stop time.

int sources
A bit mask with a bit set for all trigger sources that can trigger this channel. The gates cannot use the additional digital trigger sources NDIGO_TRIGGER_SOURCE_TDC_PE to NDIGO_TRIGGER_SOURCE_BUS3_PE.

```
#define NDIGO_TRIGGER_SOURCE_A0      0x00000001
#define NDIGO_TRIGGER_SOURCE_A1      0x00000002
#define NDIGO_TRIGGER_SOURCE_B0      0x00000004
#define NDIGO_TRIGGER_SOURCE_B1      0x00000008
#define NDIGO_TRIGGER_SOURCE_C0      0x00000010
#define NDIGO_TRIGGER_SOURCE_C1      0x00000020
#define NDIGO_TRIGGER_SOURCE_D0      0x00000040
#define NDIGO_TRIGGER_SOURCE_D1      0x00000080
#define NDIGO_TRIGGER_SOURCE_TDC     0x00000100
#define NDIGO_TRIGGER_SOURCE_GATE    0x00000200
#define NDIGO_TRIGGER_SOURCE_BUS0    0x00000400
#define NDIGO_TRIGGER_SOURCE_BUS1    0x00000800
#define NDIGO_TRIGGER_SOURCE_BUS2    0x00001000
#define NDIGO_TRIGGER_SOURCE_BUS3    0x00002000
#define NDIGO_TRIGGER_SOURCE_AUTO    0x00004000
#define NDIGO_TRIGGER_SOURCE_ONE     0x00008000
```

3.4.5 Structure ndigo_extension_block

The extension block is not used in the Ndigo250M.

3.4.6 Run Time Control

int ndigo_start_capture(ndigo_device *device)

int ndigo_pause_capture(ndigo_device *device)

int ndigo_continue_capture(ndigo_device *device)
Call this to resume data acquisition after a call to ndigo_pause_capture.

int ndigo_stop_capture(ndigo_device *device, int channel_mask)

int ndigo_single_shot(ndigo_device *device, int channel_mask)
Enable the selected channels. A channel is disabled again immediately after the first trigger on that channel.

int ndigo_manual_trigger(ndigo_device *device, int channel_mask)
Cause the selected channels to trigger immediately. They must be enabled.

```
#define NDIGO250M_CHANNEL_A_MASK    0x01
#define NDIGO250M_CHANNEL_B_MASK    0x02
#define NDIGO250M_CHANNEL_C_MASK    0x04
#define NDIGO250M_CHANNEL_D_MASK    0x08
#define NDIGO250M_CHANNEL_T_MASK    0x10
```

3.5 Readout

int ndigo250m_read(ndigo_device *device, ndigo250m_read_in *in, ndigo250m_read_out *out)
For each DMA channel returns a pointer to an array of captured data in read_out. The result can contain any number of packets of type ndigo_packet. read_in provides parameters to the driver. A call to this method automatically allows the driver to reuse the memory returned in the previous call.

Returns an error code as defined in the structure ndigo_read_out.

int ndigo250m_acknowledge(ndigo_device *device, ndigo_packet *packet)
Acknowledge all data up to the packet provided as parameter. This is mandatory if acknowledge_last_read in the ndigo_read_in structure is set to false for calls to ndigo_read.

This feature allows to either free up partial DMA space early if there will be no call to ndigo_read anytime soon. It also allows to keep data over multiple calls to ndigo_read to avoid unnecessary copying of data.

int ndigo_process_tdc_packet(ndigo_device *device, ndigo_packet *packet)
Call on a TDC packet to update the timestamp of the packet with a more accurate value. If called more than once on a packet the timestamp will be invalid.

3.5.1 Input Structure ndigo250m_read_in

ndigo_bool_t acknowledge_last_read
If set ndigo_read automatically acknowledges packets from the last read.

mask
This is a bitmask that defines which channels are to be read. All other channels will be left untouched. A value of 0 reads all channels.

3.5.2 Input Structure ndigo250m_read_out

ndigo_packet *first_packet[NDIGO25M_DMA_COUNT]
For each DMA channel a pointer to the first packet that was captured by the call of ndigo_read.

ndigo_packet *last_packet[NDIGO25M_DMA_COUNT]
For each DMA channels the address of the header of the last packet in the buffer.

int error_code[NDIGO25M_DMA_COUNT]
#define NDIGO_READ_OK 0
#define NDIGO_READ_NO_DATA 1
#define NDIGO_READ_INTERNAL_ERROR 2

const char *error_message[NDIGO25M_DMA_COUNT] Error messages created by readout

separately for each channel.

3.6 Other Functions

3.6.1 LED control

There are six LEDs on the front panel. The intensity of the red and green part can be set from 0 to 255. There is no blue component in the current version. Per default all LEDs are set to auto mode. This means that used channels are lit green, activity is shown as yellow on overflow is shown as red.

int ndigo_set_led_color(ndigo_device *device, int led, unsigned short r, unsigned short g, unsigned short b)
Set the LED to the selected color. No automatic updates are performed.

int ndigo_set_led_automode(ndigo_device *device, int led)
Let the selected LED be controlled by hardware.

3.7 Packet Format

3.7.1 Output Structure ndigo_packet

unsigned char channel
0 to 3 for the ADC input channels, 4 for the TDC, 5 for the timestamp channel.

unsigned char card
Identifies the source card in case there are multiple boards present. Defaults to 0 if no value is assigned to the parameter board_id in Structure ndigo_init_parameters or set via
int ndigo_set_board_id(ndigo_device *device, int board_id).

unsigned char type
For the ADC channels this is set to 1 to signify 16 bit signed data.

For the TDC channel it is set to 8 to signify 64 bit unsigned data.

If the type field is 128 or greater then there is no data present, even if length is not 0. In this cases the length field may contain other data.

Type	Length Field	Description
1	Number of payload words	16 bit signed samples from one of the ADCs
8	Number of payload words	64 Bit unsigned TDC Data, only for internal processing
128	Bit pattern of trigger sources	Whenever at least one of the sources that is enabled for the timestamp channel triggers, one of these packets is generated. The length field contains the triggers that are active when this packet was created.

unsigned char flags

If the bit with weight 1 is set, the packet was truncated because the internal FIFO was full.

If the bit with weight 2 is set, there are lost triggers immediately preceding this packet due to insufficient DMA buffers.

If the bit with weight 4 is set, the packet contains ADC sample overflows.

If the bit with weight 8 is set, there are lost triggers immediately preceding this packet due to insufficient buffers.

unsigned int length
Number of 64-bit elements (each containing 4 samples) in the data array if type < 128.

If type $= 128$ this is the pattern of trigger sources that where active in the clock cycle given by the timestamp. Bits are set according to the trigger sources, i.e. bit 0 is set if trigger A0 was active, bit 29 is set if trigger BUS3_PE was active. Use the NDIGO_TRIGGER_SOURCE_* # defines to check for the bits set.

unsigned __int64 timestamp
ADC channels A to D: Timestamp of the last word in the packet in ps.

TDC: Timestamp of the trigger event (falling edge) on the TDC channel in ps. When ndigo_process_tdc_packet() is called once on the packet the timestamp is replaced with the precise timestamp for the edge.

Timestamp channel: Timestamp of the trigger event in ps.

unsigned __int64 data[]
Sample data. For the Ndigo5G each 64 bit word contains four 16 bit signed words from the ADC. The user can cast the array to short* to directly operate on the sample data.

4 C Example

```c
#include "Ndigo_interface.h"
#include "Ndigo250M_interface.h"
#include <stdio.h>
#include <stdlib.h>

#define BUFFER_SIZE (1 << 21)    //     2 MB

int main(int argc, char* argv[])
{
  ndigo_init_parameters params;
  ndigo250m_get_default_init_parameters(&params);

  params.card_index = 0;
  for (int i = 0; i < NDIGO250M_DMA_COUNT; i++)
    params.buffer_size[i] = BUFFER_SIZE;


  int error_code;
  const char *error_message;
  ndigo_device* ndgo = ndigo250m_init(&params, &error_code, &error_message);
  if( error_code != NDIGO_OK ) {
    printf("\nError %d: %s\n", error_code, error_message);
    exit(-1);
  }

  ndigo250m_configuration config;
  ndigo250m_get_default_configuration(ndgo, &config);


  // disable unused trigger blocks
  config.trigger_block[1].enabled = false;
  config.trigger_block[2].enabled = false;
  config.trigger_block[3].enabled = false;
  config.trigger_block[4].enabled = false;

  // configure trigger block 0
  config.trigger_block[0].enabled = true;
  config.trigger_block[0].minimum_free_packets = 1.0;
```

```
39    config.trigger_block[0].precursor = 1;
40    config.trigger_block[0].retrigger = 0;
41    config.trigger_block[0].sources = NDIGO_TRIGGER_SOURCE_A0;
42    config.trigger_block[0].length = 16;
43    config.trigger_block[0].gates = NDIGO_TRIGGER_GATE_NONE;
44
45    config.analog_offset[0] = 0.1;
46
47    config.trigger[NDIGO_TRIGGER_A0].edge = true;
48    config.trigger[NDIGO_TRIGGER_A0].rising = false;
49    config.trigger[NDIGO_TRIGGER_A0].threshold = 0;
50
51    if( ndigo250m_configure(ndgo, &config) != NDIGO_OK ) {
52      printf("\nFatal configuration error. Aborting...\n");
53      exit(-1);
54    }
55
56    ndigo_start_capture(ndgo);
57
58    // counts the number of packets received
59    int count = 0;
60
61    while( count < 10 ) {
62      ndigo_read_in in;
63      // Do not wait for data
64      // (if set to 1 the ndigo_acknowledge function has to be removed)
65      in.acknowledge_last_read = 0;
66      ndigo_read_out out;
67      int result = ndigo_read(ndgo, &in, &out);
68      if( !result ) {
69        // buffer received with one or more packets
70        ndigo_packet *packet = out.first_packet;
71        while( packet <= out.last_packet ) {
72          int length = 0;
73          if( !(packet->type & NDIGO_PACKET_TYPE_TIMESTAMP_ONLY) )
74            length = packet->length;
75
76          printf("Card %d, Channel %d, Flags %d, Length %d, Timestamp %llu \↩
               n",
77            packet->card, packet->channel, packet->flags, length, packet->↩
                 timestamp);
78
79          if( !(packet->type & NDIGO_PACKET_TYPE_TIMESTAMP_ONLY) )
80          {
81            short* data = (short*) packet->data;
82            for( unsigned int i = 0; i < packet->length * 4; i++ )
83              printf("%6d, ", *(data++));
84            printf("\n\n");
85          }
```

```
86          // current packet pointer is invalid after call to ↩
                ndigo_acknowledge()
87          ndigo_packet *next_packet = ndigo_next_packet(packet);
88          ndigo_acknowledge(ndgo, packet);
89          packet = next_packet;
90          count++;
91        }
92      }
93    }
94    ndigo_close(ndgo);
95    return 0;
96  }
```

5 Technical Data

5.1 Operating Conditions

5.1.1 Environmental Conditions for Operation

The board is designed to be operated under the following conditions:

Symbol	Parameter	Min	Typ	Max	Units
	ambient temperature	5		40	°C
	relative humidity at 31°C	20		75	%

5.1.2 Environmental Conditions for Storage

The board shall be stored between operation under the following conditions:

Symbol	Parameter	Min	Typ	Max	Units
	ambient temperature	-30		60	°C
	relative humidity at 31°C non condensing	10		70	%

5.2 Digitizer Characteristics

Each board ist tested against the values listed in the 'Min' column. 'Typ' is the mean value of the first 10 boards produced.

PCIe throughput is 1400MB/s typical. (x8 Gen1)

Symbol	Parameter	Min	Typ	Max	Units
THD	Total Harmonic Distortion	72			dB
SNR	Signal to Noise Ration	62			dB
$SFDR_{incl}$	Spurious Free Dynamic Range (including Harmonics)	70			dB
$SFDR_{excl}$	Spurious Free Dynamic Range (excluding Harmonics)	90			dB
SINAD	Signal-to-Interference Ratio including Noise and Distortion	61			dB
ENOB	Effective Number of Bits	10			

5.3 TDC Characteristics

Symbol	Parameter	Min	Typ	Max	Units
BIN_{TDC}	TDC bin size				ps
INL_{TDC}	TDC integral nonlinearity				
	Input Pulse Width	3.3			ns
	Input Pulse Spacing (Deadtime)	32			ns

5.4 Electrical Characteristics

5.4.1 Oscillator

The Ndigo250M-14 uses an OCXO oscillator with 25ppb stability. After power up the oscillator needs to run for 10 minutes to reach this stability

5.4.2 Power Supply

Symbol	Parameter	Min	Type	Max	Units
I	PCIe 3,3V rail power consumption			0,4	A
VCC	PCIe 3,3V rail power supply	3,003	3,3	3,579	V
I	PCIe 12V rail power consumption			2,1	A
VCC	PCIe 12V rail power supply	11,04	12	12,96	V
I	PCIe 3,3VAux rail power consumption		0		A
VCC	PCIe 3,3VAux rail power supply		3,3		V

5.4.3 Analog Inputs

DC coupled single ended analog inputs. (AC coupled available as an option.)

Symbol	Parameter	Min	Typ	Max	Units
V_{off}	input center offset	0		2.5	V
V_{in}	Input Voltage	V_{off}-0.5		V_{off}+0.5	V
V_{p-p}	Input Voltage Peak to Peak		1		V
Z_A	input impedance		50		Ω
f_{-3dB}	-3dB frequency	90	92	100	MHz
f_{-6dB}	-6dB frequency	100	105	110	MHz

The corresponding frequency response is shown in fig. 6.1.

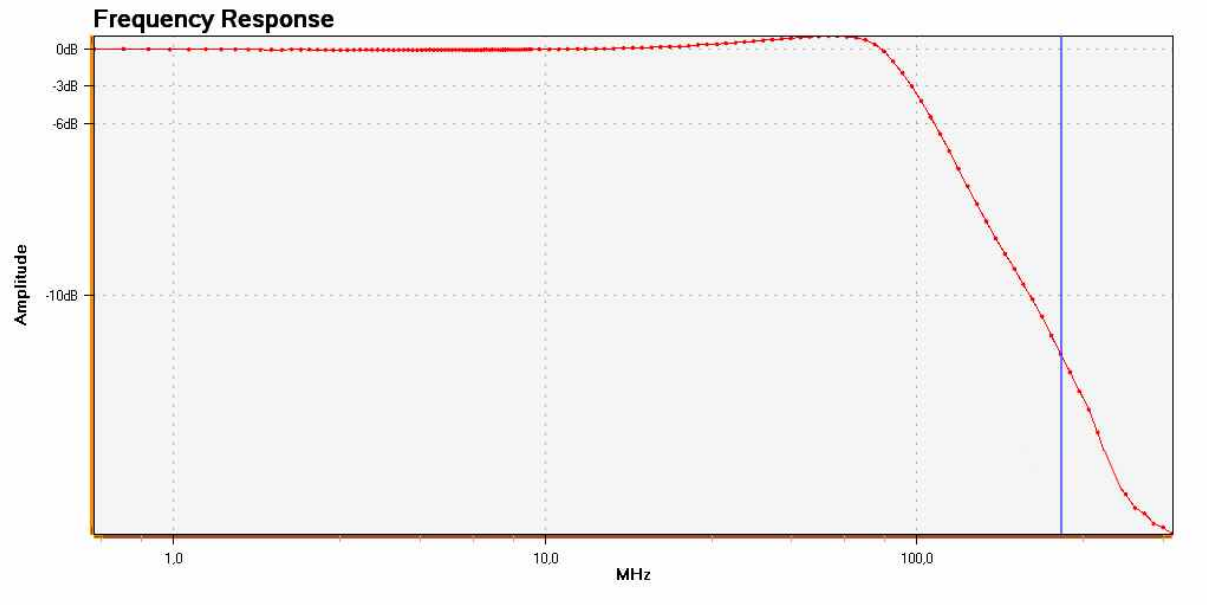

Figure 5.1: Frequency response of the tested channel.

5.4.4 Digital Inputs

AC coupled single ended digital inputs TRIGGER and GATE.

Symbol	Parameter	Min	Typ	Max	Units
V_{Dth}	discrimination threshold	-1.25		1.25	V
V_{dn}	recommended amplitude for negative pulses	-0.1 - V_{Dth}	$2V_{Dth}$	1.25 - V_{Dth}	V
V_{dn}	recommended amplitude for negative pulses	0.1 + V_{Dth}	$2V_{Dth}$	1.25 + V_{Dth}	V
Z_D	input impedance		50		Ω

5.5 Information Required by DIN EN 61010-1

5.5.1 Manufacturer

The Ndigo250M-14 is a product of:

cronologic GmbH & Co. KG
Jahnstraße 49
60318 Frankfurt

HRA 42869 beim Amtsgericht Frankfurt/M

VAT-ID: DE235184378

5.5.2 Intended Use and System Integration

The devices are not ready to use as delivered by cronologic. It requires the development of specialized software to fulfill the application of the end user. The device is provided to system integrators to be built into measurement systems that are distributed to end users. These systems usually consist of a the Ndigo250M, a main board, a case, application software and possible additional electronics to attach the system to some type of detector. They might also be integrated with the detector.

The Ndigo250M-14 is designed to comply with DIN EN 61326-1 when operated on a PCI compliant CEM main board housed in a properly shielded enclosure. When operated in a closed standard compliant PC enclosure the device does not pose any hazards as defined by EN 61010-1.

Radiated emissions, noise immunity and safety highly depend on the quality of the enclosure. It is the responsibility of the system integrator to ensure that the assembled system is compliant to applicable standards of the country that the system is operated in, especially with regards to user safety and electromagnetic interference. Compliance was only tested for attached cables shorter than 3m.

All power supplied to the system must be turned off before installing the board.

When handling the board, adequate measures have to be taken to protect the circuits against electrostatic discharge (ESD).

5.5.3 Cooling

The Ndigo250M-14 in its base configuration has passive cooling.

5.5.4 Environmental Conditions

Refer to the sections on Environmenal Conditions on page 49

5.5.5 Inputs

The analog inputs are DC coupled. The inputs have very high input bandwidth requirements and therefore there are no circuits that provide over voltage protection for these signals. Any voltage on the inputs above 5V or below -3V relative to the voltage of the slot cover can result in permanent damage to the board.

5.5.6 Recycling

cronologic is registered with the “Stiftung Elektro-Altgeräte Register” as a manufacturer of electronic systems with Registration ID DE 77895909.

The Ndigo250M-14 belongs to category 9, “Überwachungs und Kontrollinstrumente für ausschließlich gewerbliche Nutzung”. The last owner of a Ndigo250M-14 must recycle it or treat the board in compliance with §11 and §12 of the German ElektroG or return it to cronologic at the address listed on page 52.